Quick Incompact3d guide and notes

This is a quick guide on to how to use **Incompact3d** and to know some details of its functioning. The present guide and the `modified` version of the code (baseline `v4.0`) are unofficial. Please refer to the official website of the code: https://www.incompact3d.com and the major references [1],[2],[3],[4].

Compiling

In order to use *Incompact3d*, the external Fortran library used for parallelization and FFT `2decomp-fft` must be compiled. We first start by describing its installation.

Compilation of 2decomp-fft
In the folder `/5-external/2decomp-fft` create a `build` directory:

```
mkdir build
```

Before preparing the `make` file with `cmake` (version `v3.21` or higher), we need to define the variable `FC` (Fortran Compiler) of `cmake` used to compile *2decomp-fft* library:

```
export FC=ifort
```

where in this example the Intel Fortran compiler `ifort` is specified. Another common possibility is the GNU compiler `gfortran`, that is the default option. In order to use `ifort`, we need to export the `mkl` directory path (even if a different FFT implementation is used; this is needed by `Incompact3d` later due to `cmake`):

```
export MKL_DIR=${MKLROOT}/lib/cmake/mkl
```

Now, it is possible to define the `make` file and (optionally) to select the specific FFT implementation (`FFT_Choice` parameter). Options available are: `generic, fftw, fftw_03, mkl`. Default option is `generic`.

The `generic` FFT implementation is based on an algorithm that is attributed to Glassman (refer to Glassman's general N Fast Fourier Transform). It is not particularly efficient, but serves two purposes:

- It makes the package independent to external libraries, aiding portability.
- It takes over the computation if the external FFT engine fails to work (for example, if a user mistakenly passes in an input with length not supported by the underlying FFT engine).

It is not recommended to use this FFT engine in production works as it may lose 2 digits of accuracy when running in double-precision mode. This explanation on FFT `generic` is taken from https://2decomp-fft.github.io/.

The `mkl` option (Intel Math Kernel Library) seems to be the best option for 3D FFT (Gambron & Thorne, 2020). However, this FFT implementation seems to give problems on small domains (this needs further checks). In the *build* directory we can run:

```
cmake -DFFT_Choice=mkl ../
```

It is also possible to specify single precision for saving fields, through the `cmake` option `SINGLE_PRECISION_OUTPUT`. This allows to maintain double precision for calculations and for checkpoint files. The full `cmake` command is thus:

```
cmake -DFFT_Choice=mkl -DSINGLE_PRECISION_OUTPUT=ON ../
```

In closing the `cmake` command options, we remind that is possible to directly specify the compiler choice during `cmake`, through the variable `CMAKE_Fortran_COMPILER`. For example: `-DCMAKE_Fortran_COMPILER=ifort`. Moreover, it is possible to clean the `cmake` cache (to avoid to delete and create again the build folder) with the option `--fresh` added in the full `cmake` command.

After that, the *makefile* is generated. In the same folder we can compile:

```
make -j n
```

where `n` is the number of processors employed. Finally, it is possible to install `2decomp-fft`:

```
make install
```

For further information on `2decomp-fft` please refer to the original documentation, available at: https://github.com/2decomp-fft.

Compilation of Incompact3d
Similarly to the `2decomp-fft` installation, we create a `build` directory under `/2-solver`:

```
mkdir build
```

We then define the make file through `cmake` inside the `build` directory. The default Fortran compiler is `gfortran` as *2decomp-fft*; the exported `cmake` variable `FC` is valid also here. The `FC` variable can be unset through `unset FC`. Floating-point precision for outputs must be specified also in this case (if needed):

```
cmake -DSINGLE_PRECISION_OUTPUT=ON ../
```

We can finally compile the binary file:

```
make -j n
```

Basic functioning

In this section, the main features of the code that can be useful to run simulations are reported.

1. The code is designed to solve non-dimensional Navier-Stokes equations, since the only input parameter is Re. However, consistency of other parameters (e.g. inlet velocity and domain dimensions) allow to have dimensional simulations. The Re specified in the input file is used to compute the kinematic viscosity as

   $$\nu = \frac{1}{Re}$$

   so we are assuming unitary reference length and velocity scales. Reference scales depend then on the specific flow case.

   Channel flow
   For a channel flow, the default condition of simulation is to enforce a constant flow rate (CFR), while constant pressure gradient (CPG) can be enabled if necessary. Both conditions are rescaled with the centerline Reynolds number of the corresponding laminar Poiseuille flow, $Re_0 = \frac{U_0 h}{\nu}$, where U_0 is the centerline velocity and h is the channel half-height. A relation is available in order to estimate the related friction Reynolds number Re_τ:

   $$Re_\tau \approx 0.116 Re_0^{0.88}$$

   With CPG option enabled, the same relation is used to impose the same non-dimensionalization, since Re_τ must be specified in the input file instead of Re_0:

$$Re_0 \approx \left(\frac{Re_\tau}{0.116}\right)^{1/0.88}$$

In CFR conditions, being U_0 constant and unitary, this means that the bulk velocity U_B must be kept to the value of $2/3$. This can be demonstrated being the laminar profile parabolic.

Finally, in order to estimate the related bulk Reynolds number $Re_B = \frac{2U_B h}{\nu}$, the following relation can be employed (Pope, "Turbulent Flows"):

$$Re_\tau \approx 0.09 Re_B^{0.88} \Rightarrow Re_B \approx \left(\frac{Re_\tau}{0.09}\right)^{1/0.88}$$

Temporal Turbulent Boundary Layer (TTBL)
For a TTBL, the Reynolds number is based on the *trip wire diameter*, D and the wall velocity U_w:

$$Re_D = \frac{U_w D}{\nu}$$

A value of $Re_D = 500$ allows for the quickest transient of the initial conditions (see the main reference Kozul et al. (2016)).

2. Pressure field is made non-dimensional with the reference specific kinetic energy content $q_{ref} = \frac{1}{2}\rho U_{ref}^2$, where U_{ref} is the reference velocity of the case. The code performs operations on the pressure field with a factor Δt, but it is rescaled before saving the snapshots with `rescale_pressure` subroutine.
3. To evaluate the parameter β used to stretch the mesh elements in y direction, the Python program `mesh_evaluation.py` can be used. This script can handle pre-processing of Temporal Turbulent Boundary Layers (TTBLs) and of Channel flows. As a general indication, low β values correspond to a strong stretching of the mesh, while high β values correspond to almost uniform mesh.
4. Variables are saved on y_p points, that are the faces of the grid elements, while y_{pi} points are the centers of the grid elements (i : internal).
5. Boundary values of velocity are specified through the b_{ijk} variables, where i is the wall-normal direction of the boundary, j is the direction of the specific velocity component and k specifies if we are considering the bottom or the top walls (e.g. b_{yx1} refers to the x velocity component, specified at the bottom boundary with normal direction y).
6. Velocity boundary conditions are specified in the input file `input.i3d` with the following variables: `nclx1, nclxn, ncly1, nclyn, nclz1, nclzn`, that specify the normal direction to the boundary and if we are considering the first or the last element along the specific direction. Values that can be adopted are: 0, for *periodic BC*, 1 for *free-slip BC* and 2 for *Dirichlet BC* (so imposed velocity value). Boundary conditions can be different along the same direction, so different combinations can be enforced. Due to the nature of the code, pressure boundary conditions are not needed.
7. Boundary conditions for scalar fields have the same functioning of the velocity BCs. They are called: `nclxS1, nclxSn, nclyS1, nclySn, nclzS1, nclzSn`.
8. The hyperviscous option for the second order derivative is a way to increase the numerical dissipation of the standard 6th order accurate scheme (that is sub-dissipative). In this manner, it is possible to increase the modified wavenumber at high wavenumbers, thus increasing the *dissipation error* E_{diss}, similarly to what is observed in high-order upwind schemes. This approach allows to prevent *wiggles* and thus to improve stability, even at high cell Reynolds number (or numerical Péclet) $Pe \approx 200$. The two parameters available can be used also to control the numerical dissipation in the context of ILES simulations.
9. Implicit time integration is available for the diffusive terms in y direction. From preliminary analyses, it appears that it allows to drop the restriction due to the stability parameter $S < 1$ of fully explicit time integration schemes (Thompson et al. (1985)). The stability parameter S can be calculated in the three directions

$$S_i = \frac{u_i^2 \Delta t}{2\nu}$$

considering $i = x, y, z$. With implicit diffusion for y direction terms, the hyperviscous operator for the second order derivative can be used only with Dirichlet conditions on both boundaries. Semi-implicit time integration has been improved in order to account for non-zero velocity at the walls (translating walls: e.g. TTBL and spanwise wall oscillations) and for the use of Runge-Kutta 3 for the explicit portion of the time-integration (only Adams-Bashforth schemes were available with semi-implicit diffusion).

10. Avoid to cancel the latest `restart.info` file, in order to being able to correctly calculate the current time unit (since the current time unit before restart is read from `restart.info` file itself, that is located in the `data/restart_info` folder).
11. The total shear velocity is calculated as:

$$u_\tau = \sqrt{\nu \left(\frac{\partial U_\parallel}{\partial y} \right)_w}$$

where $\left(\frac{\partial U_\parallel}{\partial y} \right)_w = \sqrt{\left(\frac{\partial U}{\partial y} \right)_w^2 + \left(\frac{\partial W}{\partial y} \right)_w^2}$ is the total velocity gradient at the wall. This shear velocity is used to check numerical resolutions (grid spacings and viscous time). Similarly, the streamwise and spanwise shear velocities are: $u_{\tau x} = \sqrt{\nu | \frac{\partial U}{\partial y} |_w}$ and $u_{\tau z} = \sqrt{\nu | \frac{\partial W}{\partial y} |_w}$. The streamwise one is used in post-processing to rescale flow statistics.

12. The (streamwise) friction coefficient c_f is calculated as:

$$c_f = 2 \left(\frac{u_{\tau x}}{U_{ref}} \right)^2$$

where U_{ref} is the reference velocity according to the specific flow case (U_w for a TTBL and U_B for a channel).

13. The restart procedure of the code does not compromise the order of accuracy of the time-integration, since old time steps used for Adams–Bashforth schemes are stored in the `checkpoint` files. Moreover, the floating-point precision of the `checkpoint` files is the same as the precision used for running the code (single or double) and it is not altered by the compilation parameter `SINGLE_PRECISION_OUTPUT`.
14. The parameter `ivisu = 1` is used to save case-specific field for visualization. At the moment, both Channel and TTBL save the Q-criterion.

Visualization

Visualization of the results can be performed by opening the `.xdmf` snapshot files of the folder `data` with a visualization/post-processing software (e.g. Paraview). Moreover, it is also possible to visualize 2D instantaneous z-normal planes of the scalar field (if present) and x-normal planes of streamwise vorticity. This can be done by opening `.xdmf` files that can be found inside the `data/planes` folder. This can be useful for large cases, being a single plane much less memory-demanding. The frequency of saving of the planes can be set with the parameter `ioutput_plane` in the input file.

Power input for TTBLs

The power input P_{in} for a TTBL with fixed walls is calculated in the following manner (assuming $\rho = 1$):

$$P_{in,fw} = \tau_{wx} U_w = \mu \frac{\partial U}{\partial y} U_w = u_{\tau x}^2 U_w$$

where $u_{\tau,x}$ is the streamwise shear velocity. For the oscillating walls case, the additional power input required to move the wall in z direction is considered:

$$P_{in,ow} = \tau_{wx} U_w + \tau_{wz} W_w = \mu \frac{\partial U}{\partial y} U_w + \mu \frac{\partial W}{\partial y} W_w = u_{\tau x}^2 U_w + u_{\tau z}^2 W_w$$

Spanwise wall oscillations

Spanwise wall oscillations are calculated in the following manner:

$$w_w(t) = A \sin\left(\frac{2\pi}{T} t + \phi_{in}\pi\right)$$

where A is the amplitude, T is the period and ϕ_{in} is the initial phase of oscillation, given as fraction of π. Wall oscillations are enabled by setting `iswitch_wo = 1`. Parameters for wall oscillations are read from the input file. If feedback control is not enabled, A and T are in external units, while in case of closed-loop control (`ifeedback_control = 1`), the parameters are considered as rescaled in wall-units with the run-time streamwise shear velocity $u_{\tau x}$.

Post-processing

During post-processing, the user will see the creation of the following folders:

- `/data_post`: folder where flow statistics calculated by `post_incompact3d` are stored;
- `/data_post_te`: folder where flow statistics calculated by `cf_monitoring` are stored. The abbreviation `te` stands for `t: time`, `e: evolution`, since averages are performed only for a TTBL among different flow realizations.
- `/time_evolution`: folder for saving `time_evolution.txt`, that stores main quantities of interest for the temporal evolution of a TTBL. This file is created by `cf_monitoring`.
- `/num_resolutions`: folder to store information related to numerical resolutions, both in space (grid spacings) and time (viscous time unit, Kolmogorov time scale). Files stored here are generated by `cf_monitoring` and by `plot_statistics`.

A deepening on skew-symmetric form employed by Incompact3d

According to Kravchenko & Moin (1997), the non-linear term of incompressible Navier-Stokes equations can be written in four different forms:

- **Divergence form** (also called **conservative form** according to prof. Cimarelli)

$$\frac{\partial u_i u_j}{\partial x_j}$$

- **Convective form**

$$u_j \frac{\partial u_i}{\partial x_j}$$

- **Rotational form**

$$u_j \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) + \frac{\partial}{\partial x_i}\left(\frac{1}{2} u_j u_j\right) = \vec{\omega} \times \vec{u} + \nabla \frac{u^2}{2}$$

- **Skew-symmetric form**

$$\frac{1}{2}\left(\frac{\partial u_i u_j}{\partial x_j} + u_j \frac{\partial u_i}{\partial x_j}\right)$$

  They also report that for divergence and convective forms, spectral methods are energy-conserving only if dealiasing is performed. For skew-symmetric and rotational forms, both spectral and finite-difference methods are energy-conserving even in the presence of aliasing errors. In Incompact3d, the skew-symmetric form is adopted since it allows to reduce the aliasing error while remaining energy-conserving for the spatial discretization permitted in the code.

References

1. High-order compact schemes for incompressible flows, A simple and efficient method with quasi-spectral accuracy - Laizet & Lamballais - 2009
2. Incompact3d - A powerful tool to tackle turbulence problems with up to O(10^5) computational cores - Laizet & Li - 2011
3. Xcompact3D - An open-source framework for solving turbulence problems on a Cartesian mesh - Bartholomew et al. - 2020
4. Straightforward high-order numerical dissipation via the viscous term for DNS and LES - Lamballais et al. - 2011
5. Turbulent Flows - Pope
6. The cell Reynolds number myth - Thompson et al. - 1985
7. The 2DECOMP&FFT library: an update with new CPU-GPU capabilities - Rolfo et al. - 2023
8. Comparison of several FFT Libraries in C and C++ - Gambron & Thorne - 2020
9. DNS of the incompressible temporally developing TBL - Kozul et al. - 2016
10. On the Effect of Numerical Errors in Large Eddy Simulations of Turbulent Flows - Kravchenko & Moin - 1997